|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Лабораторная работа № 10**

|  |  |
| --- | --- |
| **Тема** Алгоритм Плавающего горизонта построения трёхмерных поверхностей  **Студент** Якуба Д. В.  **Группа** ИУ7-43  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Куров А. В. |  |

Москва

2020 г.

Оглавление

[Цель работы 3](#_Toc42432847)

[Техническое задание 3](#_Toc42432848)

[Теоретическая часть 3](#_Toc42432849)

[Трёхмерная графика 3](#_Toc42432850)

[Модели трёхмерных объектов 3](#_Toc42432851)

[Удаление невидимых линий поверхности 4](#_Toc42432852)

[Алгоритм Плавающего горизонта 5](#_Toc42432853)

[Решение задачи определения видимости точек кривой 8](#_Toc42432854)

[Поиск точек пересечения кривых на интервалах 10](#_Toc42432855)

[Улучшение качества изображения поверхности 12](#_Toc42432856)

[Недостаток алгоритма 14](#_Toc42432857)

[Практическая часть 14](#_Toc42432858)

[Алгоритм Плавающего горизонта 14](#_Toc42432859)

[Программная реализация алгоритма на ЯП Python 15](#_Toc42432860)

[Пользовательский интерфейс 16](#_Toc42432861)

[Демонстрация работы алгоритма 16](#_Toc42432862)

# Цель работы

Изучение и программная реализация алгоритма Плавающего горизонта построения трёхмерных поверхностей.

# Техническое задание

Должна быть разработана программа, позволяющая осуществлять ввод пределов и шага изменения координат x, z, выбора уравнения поверхности из заранее сформированного списка, построение поверхности. Должен быть обеспечен поворот изображения (поверхности) вокруг каждой из трёх координатных осей. Система координат должна быть неподвижной. Выполнить масштабирования для обеспечения размещения исходного изображения целиком в пределах поля вывода.

Список уравнений поверхностей задаётся в отдельном модуле.

# Теоретическая часть

## Трёхмерная графика

### Модели трёхмерных объектов

Модели являются отображением формы и размеров объектов. Основное назначение модели – правильно отображать форму и размеры определённого объекта.

В основном используются следующие три формы моделей:

1. Каркасная (проволочная) модель. В данной модели задаётся информация о вершинах и рёбрах объекта. Это одна из простейших форм задания модели, но он имеет один существенный недостаток: модель не всегда правильно передаёт представление о форме объекта.

2. Поверхностные модели. Это тип модели, который часто используется в Компьютерной Графике. Поверхность может описываться аналитически, либо задаваться другим способом (допустим, отдельными участками поверхности, задаваемыми в качестве участков поверхности того или иного вида). При этом вложенные криволинейные поверхности можно представлять в упрощённом виде, выполняя, например, полигональную аппроксимацию: такая поверхность будет задаваться в виде поверхности многогранника.

Данная форма также имеет свой недостаток: отсутствует информация о том, с какой стороны поверхности находится материал, а с какой стороны пустота. Это важно знать в том случае, если мы занимаемся компьютерным моделированием трёхмерных объектов.

Данный способ распространён в Компьютерной Графике, так как в данной стези нас не интересует, с какой стороны находится материал, мы оперируем поверхностями.

3. Объёмные модели. Отличие данной формы задания модели от поверхностной формы задания состоит в том, что в объёмных моделях к информации о поверхностях добавляется информация о том, где расположен материал. Это можно сделать путём указания направления внутренней нормали.

### Удаление невидимых линий поверхности

Решение данной задачи позволяет правильно представить то, как расположены поверхности в пространстве.

Данную задачу принято считать центральной задачей графики. Задача эта достаточно сложная и нет единого алгоритма её решения в силу того, что могут предъявляться разные требования к результату решения.

Решать поставленную задачу можно в объектном пространстве (в мировой системе координат) или в пространстве изображений (в экранных координатах). Точность решения в объектном пространстве выше, чем в пространстве изображения, так как в первом пространстве используются вещественные значения.

Можно считать, что в основе всех алгоритмов лежит сортировка объектов по глубине. Если в сцене объектов, то трудоёмкость будет оцениваться как:

Если же работа происходит в экранной системе координат, то трудоёмкость может оцениваться следующим образом:

Где – количество пикселей экрана. В этом случае от нас потребуется решении задачи о том, каким цветом высветить очередной пиксел. Для каждого пиксела потребуется найти ближайший к наблюдателю объект (найти объект с максимальным значением координаты ).

Таким образом, поскольку , то ответ на вопрос с какой системой работать ясен. Однако используя тот факт, что расположение по глубине объектов сцены изменяется только в точках, расположенных на линии пересечения поверхностей, можно сделать вывод о том, что использование пространства изображений окажется не хуже использования объектного пространства.

## Алгоритм Плавающего горизонта

Данный алгоритм предназначен для построения поверхности, заданных неявным уравнением .

Предположим, пусть имеется кусок некоторой заданной поверхности и от нас требуется изобразить её на экране.

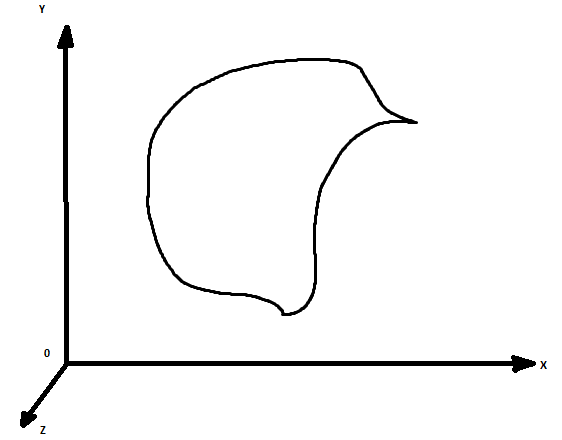


Рисунок , Пример

Наблюдатель располагается на положительной части оси и смотрит на начало координат.

Идея алгоритма состоит в том, что заданная поверхность рассекается плоскостями, перпендикулярными оси . То есть для начала от нас требуется с некоторым шагом расставить эти рассекающие плоскости.

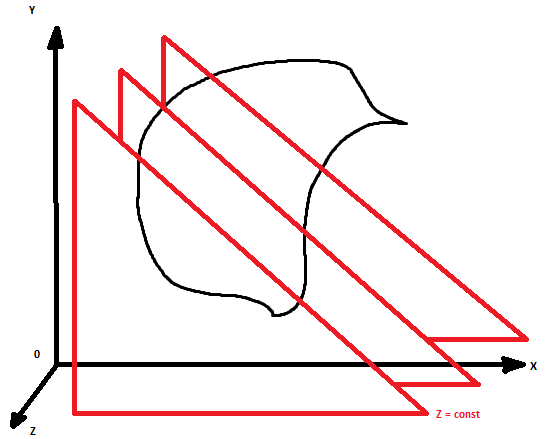


Рисунок , Рассечение

В отсечениях криволинейной поверхности и плоскостей будут получены некоторые кривые.

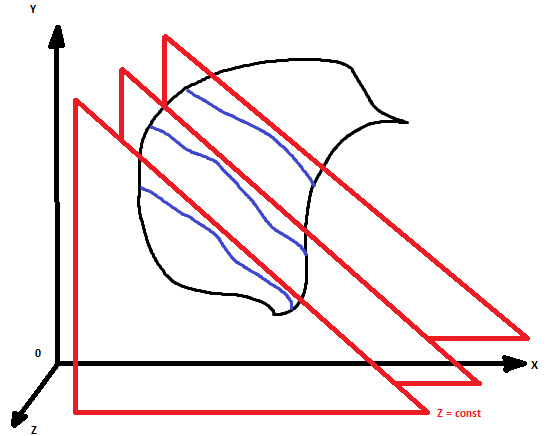


Рисунок , Результат отсечения

Зафиксируем основные этапы:

1. Рассматриваемая поверхность рассекается плоскостями, перпендикулярными оси .

В каждом отсечении получается кривая. Эта кривая, описываемая уравнением или .

2. Полученные кривые можно проецировать на плоскость (то есть на плоскость ) и изобразить видимые части каждой кривой.

Изображение строится начиная с кривой, полученной в ближайшем к наблюдателю сечении.

Кривая, полученная в сечении ближайшей плоскостью, является видимой.

Кривая, полученная во втором сечении, тоже будет видима. Связано это с тем, что вторая кривая расположена либо выше первой, либо ниже первой. В частом случае, когда они будут совпадать, будет получена одна кривая.

Начиная с третьей кривой понадобится решать задачу определения видимости точек кривой.

## Решение задачи определения видимости точек кривой

Поставленная задача решается в пространстве изображения.

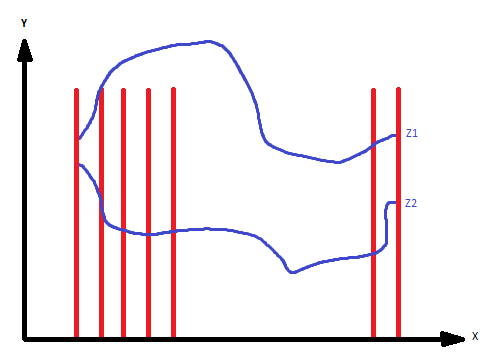


Рисунок , Пример

Берётся третья кривая и потребуется изображать те участки кривой, которые будут являться видимыми.

Участок кривой будет видим в том случае, если эта кривая располагается выше верхней кривой, либо ниже нижней кривой.

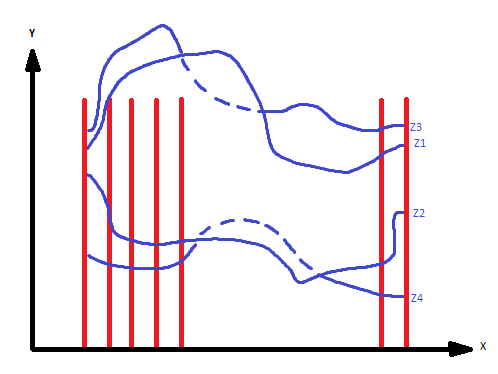


Рисунок , Добавление двух кривых с отображённой видимостью

Для того, чтобы решить поставленную задачу, потребуется вычислить функцию в очередной точке, расположенной на кривой, и сравнить значение вычисленной координаты с значением и для раннее рассмотренных кривых. В том случае, если , то очередная точка, расположенная на текущей рассматриваемой кривой, будет являться видимой. Также в случае, если , то очередная точка, расположенная на текущей рассматриваемой кривой, будет являться видимой. Если точка видима, то она высвечивается, невидима – не высвечивается.

Следует заметить, что в данном алгоритме формируются два горизонта: нижний и верхний. Верхний горизонт – видимые участки кривых с наибольшим значением ординат. Нижний горизонт – видимые участки кривых с наименьшим значением ординат.

Таким образом, требуется хранить текущее значение максимума и минимума, получаем новое значение ординаты точки, расположенной на рассматриваемой кривой, и сравниваем текущий максимум с этим значением. Если значение больше максимума, оно заносится в максимум. Если же значение не превосходит максимум, то значение сравнивается с минимумом. Если значение меньше минимума, оно заносится в минимум. Таким образом, при реализации алгоритма потребуется формировать массив максимальных и минимальных значений ординаты.

Так, верхним горизонтом можно называть участки кривых с наибольшим значением ординат, а нижним горизонтом – участки кривых с минимальным значением ординат.

Также верхним горизонтом можно назвать массив максимальных значений ординат, а нижним – массив минимальных значений ординат.

Работа алгоритма сводится к вычислению значения и сравнению если , то . В противном случае надо проверить , то . Эта процедура называется поддержанием горизонта. Точка будет невидима в том случае, если , то есть если точка расположена между двумя горизонтами.

## Поиск точек пересечения кривых на интервалах

Рассмотрим следующий пример:

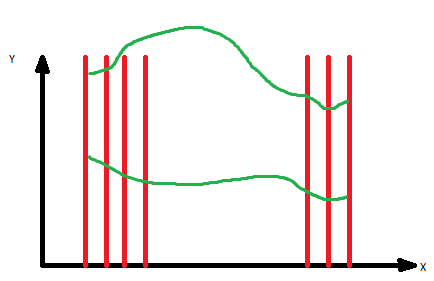


Рисунок , Пример

На рисунке зелёным цветом обозначены некоторые определённые горизонты.

Обрабатывая некоторую кривую, найдём следующие точки, ей соответствующие (обозначены синим цветом):

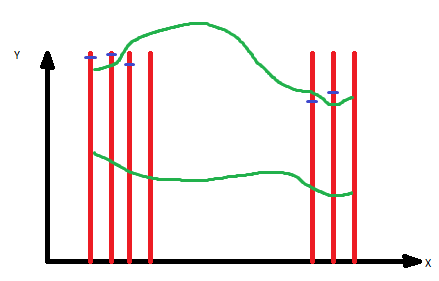
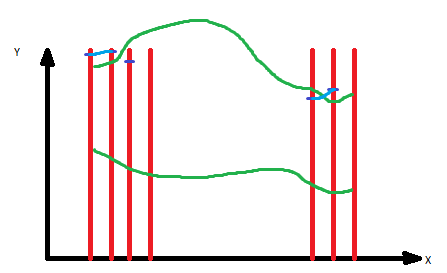


Рисунок , некоторые определённые точки

В решении задачи фигурирует следующее правило:

Если очередная точка невидима, то мы её не соединяем с предыдущей точкой. В ином случае – соединяем с предыдущей точкой. Как итог, будем иметь следующее (голубым цветом обозначена видимая часть отрезка):



Рисунок

Таким образом, предоставим на рассмотрение два ошибочных случая. В первом случае видимый участок кривой объявляется невидимым, а во втором – невидимый участок кривой объявляется видимым.

Для того, чтобы избавиться от такого недостатка и сократить и упростить вычисления, с учётом того, что достаточно мало, а также предполагая, что поверхности, а следовательно, и получаемые кривые, достаточно гладкие, на текущем интервале будем аппроксимировать кривую отрезком.

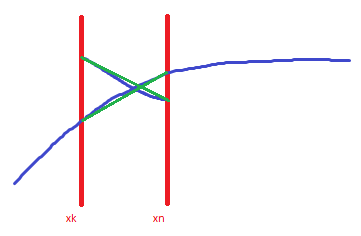


Рисунок , пример аппроксимации кривых отрезками (зелёный цвет) на интервале

На рисунке кривая, что больше, - предыдущая, а малая – текущая.

От нас требуется вычислить тангенс угла наклона каждой прямой, которая аппроксимирует эти две кривые:

Далее можем воспользоваться уравнением прямой: . Приравняем ординаты точки пересечения:

Таким образом, если текущая точка кривой невидима, то изображается участок кривой от предыдущей точки до найденной точки пересечения. Если текущая точка кривой видима, то изображается участок кривой от найденной точки пресечения до текущей точки.

## Улучшение качества изображения поверхности

Для того, чтобы рассмотреть поверхность, потребуется осуществить её поворот. Если мы поворачиваем кривые, то можно столкнуться со следующей ситуацией:

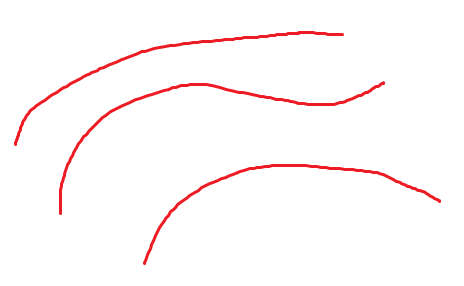


Рисунок , Ситуация

При первом взгляде на данную картинку человеческий глаз может указанную совокупность кривых не воспринимать в качестве поверхности. Качество изображения улучшится в том случае, если начальные и конечные точки каждой кривой соединить друг с другом, например, отрезками.

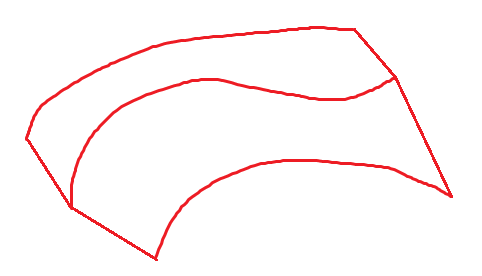


Рисунок , Улучшение качества изображения

Указанные рёбра можно соответственно назвать левым боковым и правым боковым.

Также следует отметить, что изображение можно улучшить дополнительным рассечением поверхности секущей плоскостью, перпендикулярной ещё одной оси координат. Но в данном случае нельзя построить сначала кривые, построенные в одной серии отсечении, а далее построить кривые, полученные во второй серии отсечений – построение должно происходить одновременно в каждой серии.

## Недостаток алгоритма

Рассмотрим следующую ситуацию:

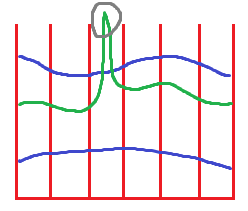


Рисунок , Демонстрация недостатка алгоритма

В результате вычисления значения функции с указанным шагом не будет вычислено значение функции в месте, указанном на рисунке серым цветом, и пик будет потерян.

Фактически это свидетельствует о том, что шаг изменения выбран неверно, и мы не сможем правильно восстановить функцию.

# Практическая часть

## Алгоритм Плавающего горизонта

Для каждой плоскости

1. Обработать левое боковое ребро.

Если точка является первой точкой кривой и лежит на первой кривой, то запомнить её (). Если точка принадлежит не первой кривой, то соединить её с точкой и запомнить в качестве новой точки .

2. Для каждой точки текущей кривой выполнить следующие действия:

Определить видимость точки (если или )

Если видимость кривой изменилась, то найти точку пересечения кривой с горизонтом.

Если текущий сегмент кривой видим, то изобразить его полностью.

Если видимость изменилась, то

Если текущая точка невидима, то изобразить участок кривой от предыдущей точки до точки пересечения

Если текущая точка видима, то изобразить участок кривой от точки пересечений до текущей точки

Заполнить массивы верхнего и нижнего горизонтов (если точка видима).

3. Обработать правое боковое ребро.

## Программная реализация алгоритма на ЯП Python

def horizonForConstant(equation, topHorizon, bottomHorizon, xStartLimit, xEndLimit, xStep, currentZ, canvasWindow):

previousPoint = 0

for currentX in arange(xStartLimit, xEndLimit + xStep, xStep):

current = transform([currentX, equation(currentX, currentZ), currentZ])

if previousPoint:

if previousPoint[0] > current[0]:

previousPoint, current = current, previousPoint

xIncrement = current[0] - previousPoint[0]

yIncrement = current[1] - previousPoint[1]

if xIncrement > yIncrement:

pickedInc = xIncrement

else:

pickedInc = yIncrement

xIncrement /= pickedInc

yIncrement /= pickedInc

newX, newY = previousPoint[0], previousPoint[1]

for \_ in range(int(pickedInc) + 1):

if newX < 0 or newX >= width or newY < 0 or newY >= height:

break

drawableX = int(round(newX))

if newY > topHorizon[drawableX]:

topHorizon[drawableX] = newY

canvasWindow.create\_rectangle((drawableX, newY) \* 2, fill = curColorLines, outline = "")

elif newY < bottomHorizon[drawableX]:

bottomHorizon[drawableX] = newY

canvasWindow.create\_rectangle((drawableX, newY) \* 2, fill = curColorLines, outline = "")

newX += xIncrement

newY += yIncrement

previousPoint = current

def drawAllHorizons(equation, topHorizon, bottomHorizon, xStartLimit, xEndLimit, xStep, canvasWindow, zStartLimit, zEndLimit, zStep):

for currentZ in arange(zStartLimit, zEndLimit + zStep, zStep):

horizonForConstant(equation, topHorizon, bottomHorizon, xStartLimit, xEndLimit, xStep, currentZ, canvasWindow)

def drawSideRibs(zStartLimit, zEndLimit, zStep, xStartLimit, xEndLimit, equation, canvasWindow):

for currentZ in arange(zStartLimit, zEndLimit, zStep):

startPoint = transform([xStartLimit, equation(xStartLimit, currentZ), currentZ])

endPoint = transform([xStartLimit, equation(xStartLimit, currentZ + zStep), currentZ + zStep])

canvasWindow.create\_line(startPoint[0], startPoint[1], endPoint[0], endPoint[1], fill = curColorLines)

startPoint = transform([xEndLimit, equation(xEndLimit, currentZ), currentZ])

endPoint = transform([xEndLimit, equation(xEndLimit, currentZ + zStep), currentZ + zStep])

canvasWindow.create\_line(startPoint[0], startPoint[1], endPoint[0], endPoint[1], fill = curColorLines)

def floatingHorizonAlgorithm(equation, xStartLimit, zStartLimit, xEndLimit, zEndLimit, xStep, zStep, canvasWindow):

clearImage(canvasWindow)

topHorizon = []

bottomHorizon = []

for \_ in range(width + 1):

topHorizon.append(0)

bottomHorizon.append(height)

drawAllHorizons(equation, topHorizon, bottomHorizon, xStartLimit, xEndLimit, xStep, canvasWindow, zStartLimit, zEndLimit, zStep)

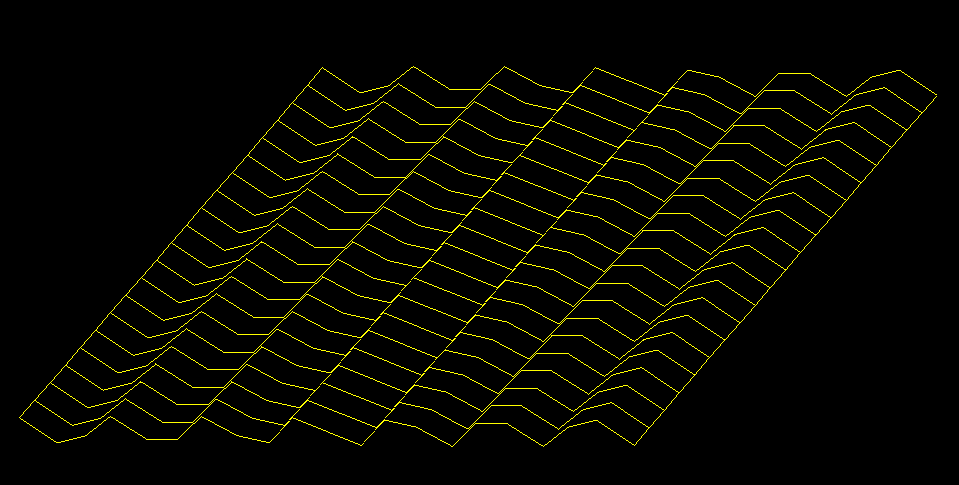
drawSideRibs(zStartLimit, zEndLimit, zStep, xStartLimit, xEndLimit, equation, canvasWindow)

## Пользовательский интерфейс

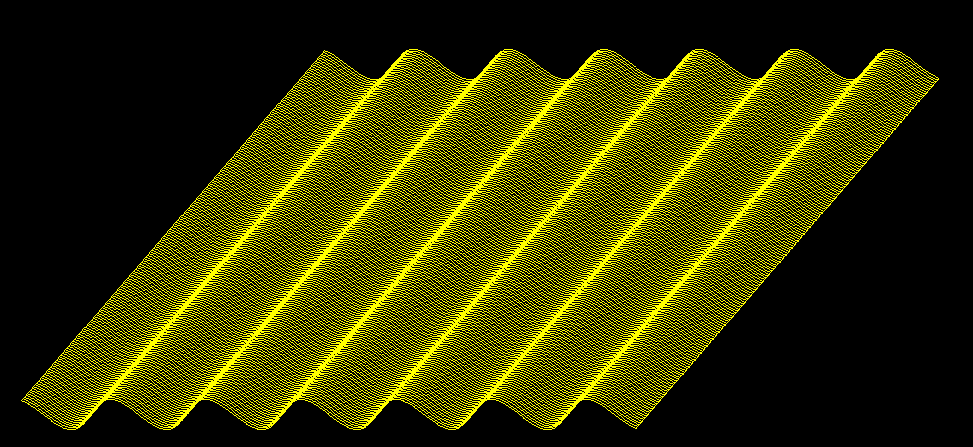


## 

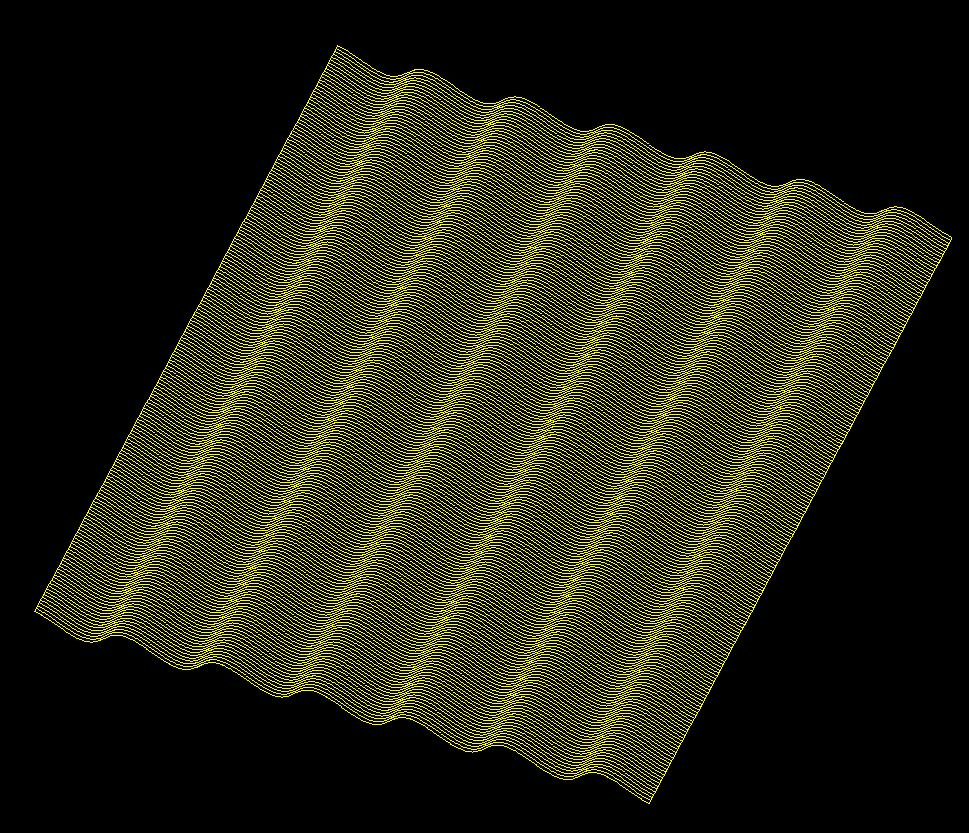
## Демонстрация работы алгоритма



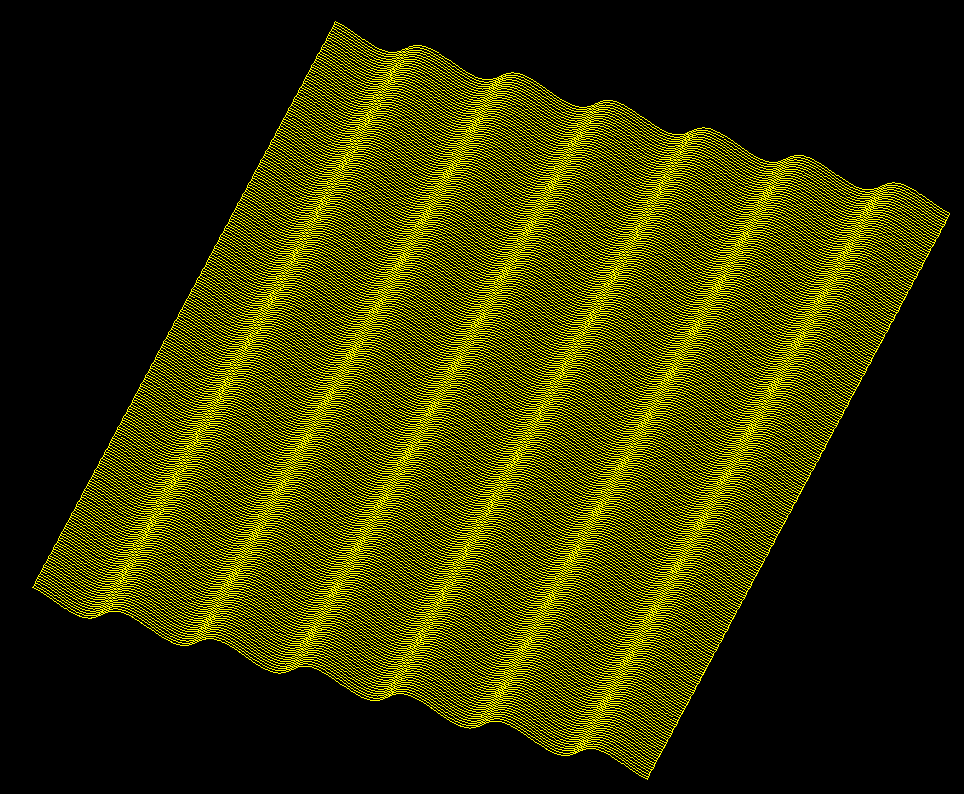
С меньшим шагом:

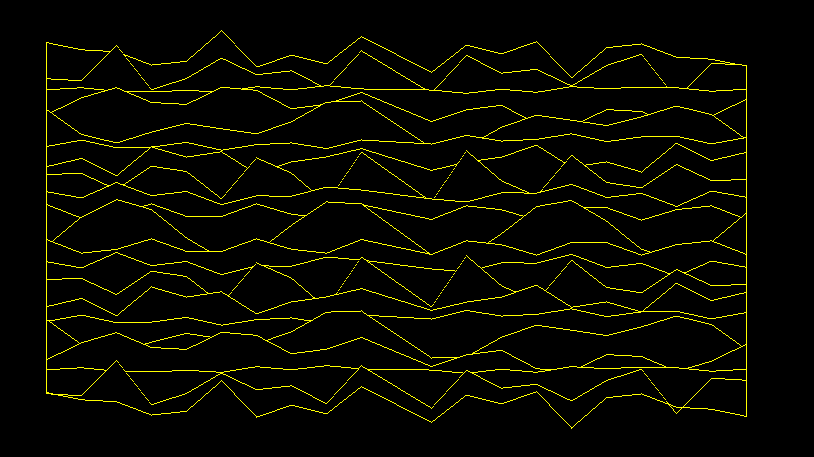


Повернём плоскость:

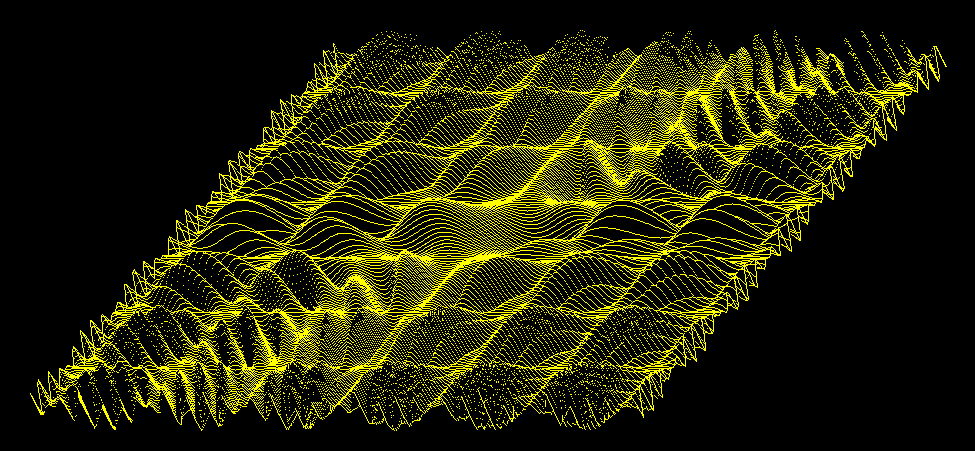


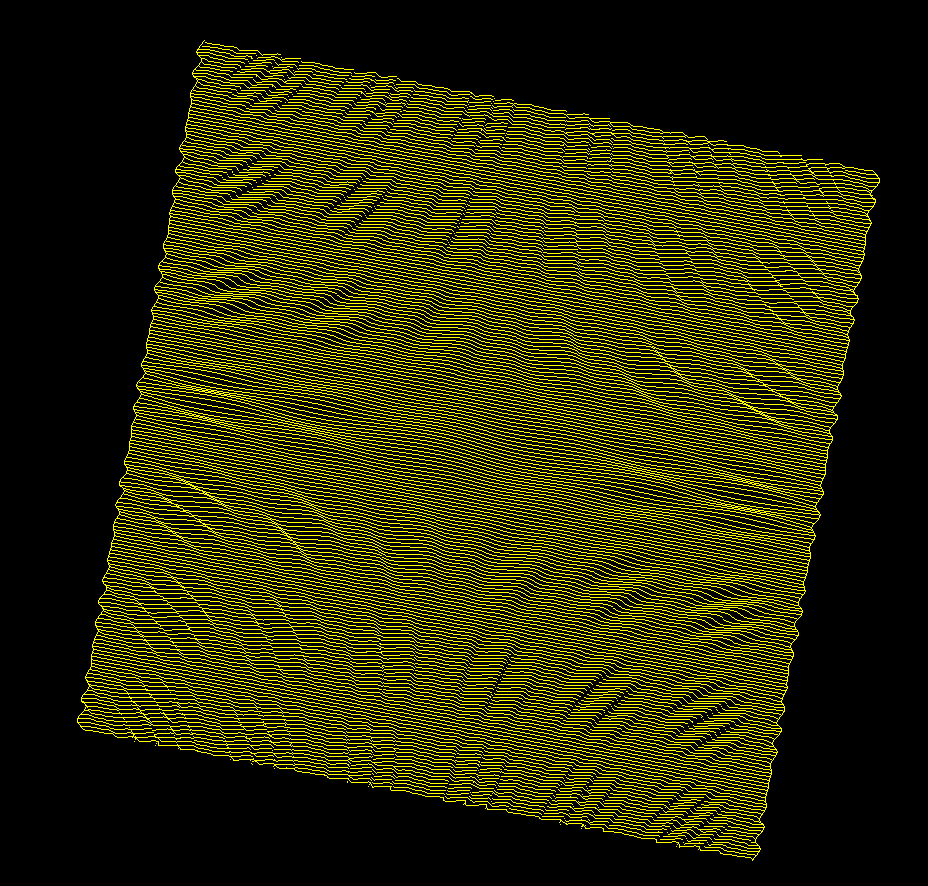
И вновь уменьшим шаг:

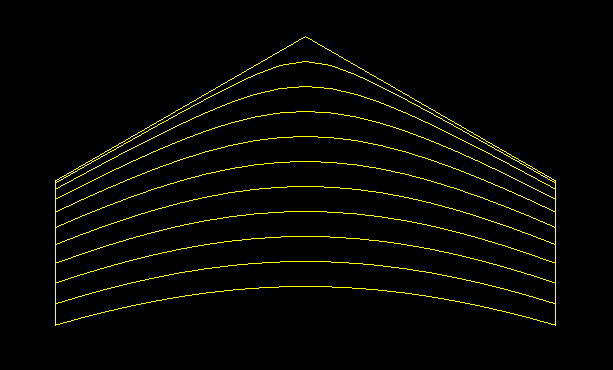


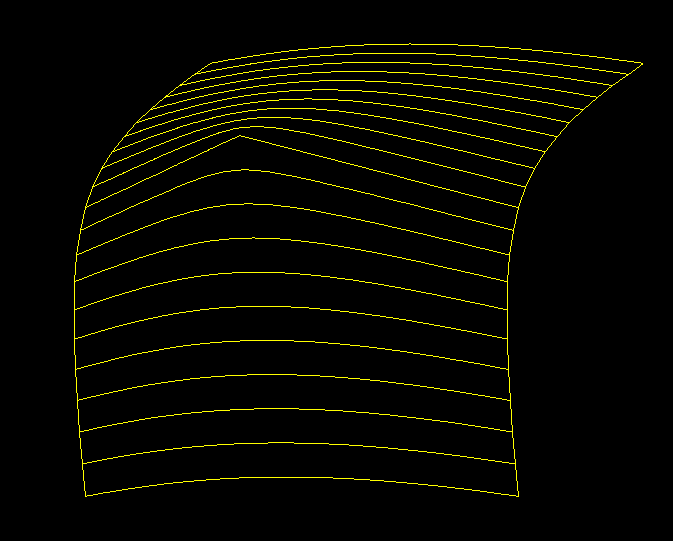


С уменьшенным шагом:

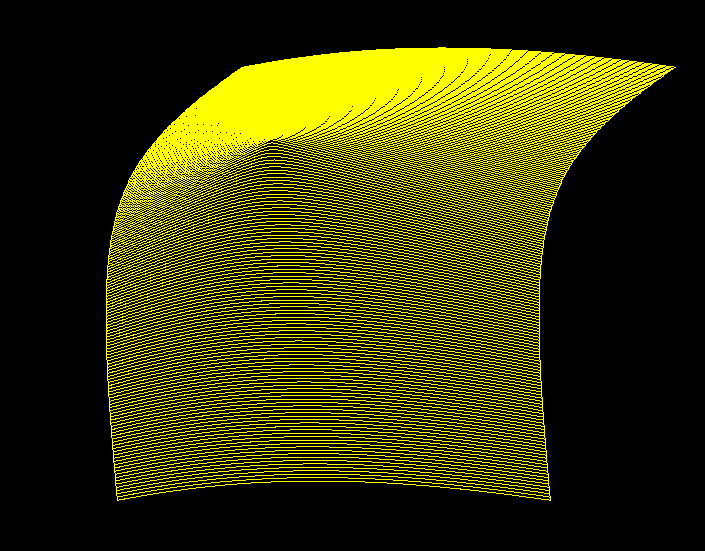


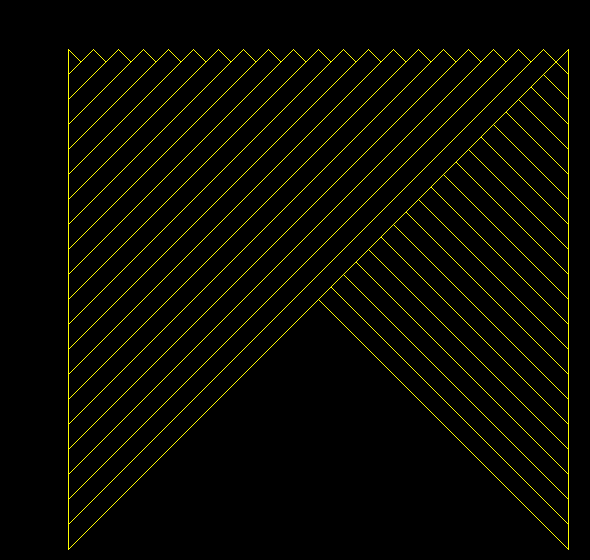


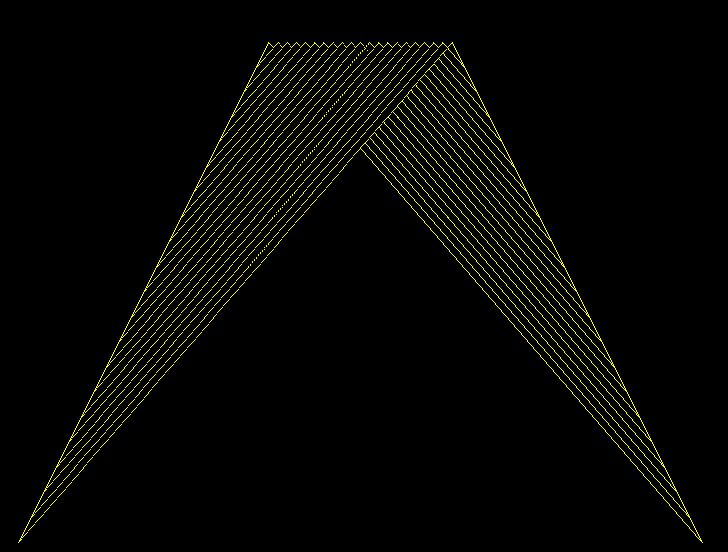
**

**

С уменьшенным шагом:







С уменьшенным шагом:

